DOCUMENT RESUME

ED 232 571 HE 016 495

AUTHOR Sholtys, Phyllis A.
TITLE Systems Prototyping with Fourth Generation Tools: One Answer to the Productivity Puzzle? AIR 1983 Annual Forum Paper.
PUB DATE May 83
NOTE 17p.; Paper presented at the Annual Forum of the Association for Institutional Research (Toronto, Canada, May 23-26, 1983).
PUB TYPE Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Case Studies; *Computer Oriented Programs; Computer Programs; Higher Education; *Information Systems; Personnel Management; Productivity; *Program Development; *Programing; Systems Approach; *User Satisfaction (Information)
IDENTIFIERS *AIR Forum; Canisius College NY

ABSTRACT

The development of information systems using an engineering approach employing both traditional programming techniques and nonprocedural languages is described. A fourth generation application tool is used to develop a prototype system that is revised and expanded as the user clarifies individual requirements. When fully defined, a combination of cost-effective techniques is used to develop the final system. Four major steps in developing a computer application are as follows: formal specification or requirements, writing the necessary computer code, testing and installing, and program documentation and user training. It is suggested that prototyping can have its greatest impact when it is used by the analyst as a tool to initially develop system specifications, rather than to test specifications produced by traditional analysis. The availability of a working model allows users to clearly understand how the system works and what information is available. In addition, users have a working system to use while a final production model is being programmed and installed. A case study of the application of prototyping for the personnel system at Canisius College illustrates this approach. (SW)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

#15

ED232571

SYSTEMS PROTOTYPING WITH FOURTH GENERATION TOOLS: ONE ANSWER TO THE PRODUCTIVITY PUZZLE?

Phyllis A. Sholtys
Director of Planning and Research
Canisius College
Buffalo, New York 14208
(716) 883-7000, Ext. 566

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

AIR

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

✓ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

NE 016 475

Paper presented at the Annual Forum of the Association for Institutional Research, Toronto, Canada, May 1983.

THE ASSOCIATION FOR INSTITUTIONAL RESEARCH

This paper was presented at the Twenty-Third Annual Forum of the Association for Institutional Research held at the Sheraton Centre in Toronto, Ontario, May 23-26, 1983. This paper was reviewed by the AIR Forum Publications Committee and was judged to be of high quality and of interest to others concerned with the research of higher education. It has therefore been selected to be included in the ERIC Collection of Forum papers.

D. R. Coleman, Chairman
Forum Publication
Advisory Committee

Abstract

Traditional systems development is extremely time consuming because analysts spend much time attempting to accurately define users' needs. End users have difficulty visualizing the needed system in the abstract and are reluctant to sign off on specifications. After implementation, their perceptions of what is needed often change and requests for changes or enhancements produce frustration for both the user and the programmer.

The development of information systems using an engineering approach employing both traditional programming techniques and nonprocedural languages is described in this paper. A Fourth generation application tool is used to quickly develop a prototype system which is revised and expanded as the user clarifies his/her requirements. When fully defined, a combination of cost-effective techniques is used to develop the final system.

At a private college where this strategy was used, significantly less development time was needed for systems development.

SYSTEMS PROTOTYPING WITH FOURTH GENERATION TOOLS:
ONE ANSWER TO THE PRODUCTIVITY PUZZLE?

Computers are continuing to increase in speed and capabilities and decrease in cost, with sales predicted to increase by 25% per year for at least the next decade (Martin, 1982). There is an inherent problem with this growth however. Computer programming has remained a labor intensive activity with only modest improvement in productivity. The rapid decrease in cost for computing power is offset by a sharp increase in the amount of money spent on program development, maintenance and enhancement and a worsening of the shortage of skilled programmers. If there is no increase in programmer productivity, it has been suggested that the current workforce of 300,000 programmers will need to grow to 28 million programmers over a period of ten years (Martin, 1982). At these rates, it would not be long before our entire worker population would be needed to program our computers! Further, the projected cost for a skilled programmer is expected to exceed $100,000 a year by the mid-80's (including salary, benefits and overhead) (Wasserman & Gutz, 1982). There is clearly an imperative to make programmers much more productive or develop alternatives to the process itself.

Within higher education, the problem is further compounded. Higher education is weathering a period of declining resources and overall instability. In coping with this environment college administrators are becoming increasingly aware of the need for improved information as a basis for planning and effective management. Department and office heads are also starting to realize that availability of timely and accurate data is essential to enhance their effectiveness. As a result, throughout our institutions there is a increasing demand for new computer applications.

A basic challenge facing individual computer centers is one of supply and demand. The demand for information systems far outstrips the supply of people to develop the systems. Programming backlogs are a way of life at most centers and their magnitude is increasing. There is clearly a need to make the software development process more effective because there is a limited supply of new resources which can be committed to the development of computer applications.

New software tools with the potential to improve systems development productivity are available but present utilization is limited. The development of information systems using an engineering approach which utilizes both traditional techniques and fourth generation tools is described in this paper. This method can be used to develop systems which provide greater user satisfaction and require less development time.

The Software Development Process

There are four major steps to be accomplished in developing any computer application: 1) formal specification of requirements; 2) writing the necessary computer code; 3) testing and installing and 4) program documentation and user training. Traditional development methodology rigidly follows each step in the sequence above. This approach is predicated on the assumption that system requirements can be completely and accurately specified at the start of a project. The remainder of the process is designed to produce a finished product that accurately follows the initial specification. However this prescribed process does not work well in many situations.

The computer field is filled with stories of systems being installed and users saying it is not what is needed. A typical reaction to these design disasters is to say that requirements had not been specified with

enough detail and to respond by using even more structured procedures to develop rigorous software specifications prior to writing the programs (Bryce, 1983; Stamen & Costello, 1981). Thus traditional systems analysis has become extremely time consuming because analysts must spend much time in attempting to accurately define users' needs. This often exceeds the time and effort required to program and complete the application . Despite the time devoted by analysts in defining specifications, end users have difficulty visualizing the needed system in the abstract and, after implementation, often decide they need something different. Once necessary changes are incorporated and users become comfortable with their system, users discover that additional functions and reports would be useful. Requests for these enhancements produce more frustration for both the user and the programmer.

Attempts to obtain exact specifications from users as a first step, are doomed to fail because there remains a basic communication problem between users and programmers. Users do not understand technical terms, specification documents or systems charts and programmers do not understand the nuances of the users' operations. The expenditure of considerable time and much frustration accompany the analysis process because participants - both user and analyst - are dealing with abstractions. What is urgently needed is a way of moving quickly from the abstract to the tangible.

Programmer Productivity

Most programming techniques and technological improvements have been aimed at decreasing the time spent on the programming and installation phases of application development. Little has been proposed that can speed up the initial analysis process.

Originally all computers were programmed in machine language, a binary code. This cumbersome and tedious task gave way to programming in assembly language. Assembly language allowed programmers to use mnemonics which are automatically translated into machine language. As computers developed, second generation languages such as COBOL, FORTRAN, and BASIC significantly increased programmer productivity by their one-to-many relationships. That is, one line of instruction in a second generation language produces a much larger number of assembly language lines. The third generation of computer programming introduced report generators, structured programming methodologies, on-line programming, data base management systems, language generators, a variety of utility programs and enhancements to second generation technology. A number of more powerful software tools for application development have now been introduced. Included in this fourth generation of software are such products as on-line relational data base management systems, interactive applications generators, non-procedural languages and integrated enhancements to third generation products.

It is claimed that the new products have the potential to dramatically improve programming productivity and initial results from sites using these tools are encouraging Many applications can be generated quickly (Doty & Krumrey, 1982; Martin, 1982; McCracken, 1980b; Rhodes, 1982; Thiel, 1982a; Wasserman & Gutz, 1982). Although much discussed, acceptance and utilization remain very limited. Professional programmers tend to resist using these new tools because 1) they are more comfortable with traditional languages and established methodologies, 2) data processing managers resist being pioneers, 3) present fourth generation applications tools cannot create every type of application that

conventional programming can, and 4) complex systems developed with fourth generation software tools often require a heavy commitment of computer resources and may be much slower to execute than comparable systems developed with conventional methods.

Even when utilized, faster programming methods still fail to address the part of the system development process where most of the problems originate - the initial design and requirements specification stage.

Alternative Strategies

One proposed way of reducing programming backlogs (and insuring that users get what they want) is to provide users with high level tools to develop their own applications. Some of the fourth generation tools have been heralded as a means whereby end users can develop their own applications without professional programmers (Johnson, 1982; Lerner, 1982; Martin, 1982; Wasserman & Gutz, 1982; among others). However this may not be the most effective utilization of either the technology or the personnel involved. Do we really want our Business Managers, Registrars, Deans, and other administrators to develop applications with little support from a data processing department? From an institutional perspective, "productivity" is total useful output compared to the resources employed. In systems development the measure of resources used must include not only hardware and programmer/analysts, but management and user time spent in developing the application. Time spent by end users in developing their own systems will be at the cost of reduced effectiveness in performing assigned institutional functions.

Few end users have the time or desire to develop their own applications, despite claims the products are "user friendly". Those users who are moving toward developing their own computer applications are

largely doing so because of frustration with the traditional development process and the long time lag before requests can be met.

Another alternative approach to software development includes the concept of prototyping as a means of decreasing development time. Prototyping is a long established procedure in product development in other fields (Martin, 1982). Few manufacturers would long survive if they used the same development process used in our computer centers, i.e., refine the product specifications on paper and go directly into production. In contrast, industry uses an engineering approach. After a proposed product is identified and initial specifications developed, a prototype is created. The prototype is used to test the design, insure that it performs the desired functions, and obtain the feedback needed to identify and make any desired adjustments before a final product is developed. A clear understanding of how the product will be used also guides selection of the most cost-effective techniques for production.

A number of experts now suggest that prototyping can be a useful enhancement to the computer systems development process (Hessinger, 1982; Martin, 1982; Thiel, 1983; Wasserman & Gutz, 1982). This engineering approach includes three steps. Identification and definition of the "product to be built" is the first step. Second, a prototype is developed to establish a working system. This leads users to a clear understanding of how the system works, what information is available, and enables users to identify any essential changes or additions. Third, the most cost effective techniques are selected and the final product is built.

Computer application prototypes have not been generally used because the cost of developing the prototype was as high as the cost of programming the final system. However, some of the fourth generation

software can generate an application in only a small fraction of the time required for traditional development (Martin, 1982, p. 30). Thus development of a system prototype is now feasible in terms of time and cost. Use of a prototype to test specifications will increase productivity by insuring that the system wanted by the user is the one that is developed by the programmer. Reprogramming and delays during the implementation stage are prevented.

Experience leads this writer to propose that prototyping should be even more dynamically utilized in the systems development process. Prototyping's greatest impact can be made when it is used by the analyst as a tool to initially develop system specifications, rather than to test specifications produced by traditional analysis. With this approach, steps one and two above become merged into one iterative process: preliminary analysis is brief and aimed at identifying basic system tasks following which a rough cut prototype is put up by the analyst and presented to the user as a simple, preliminary version. The fear of making a mistake or overlooking a vital element no longer haunts analyst or user since there is little need to define the ultimate system at the outset. At this point, the system is no longer an abstraction; the user can enter data and play with the system as a means of testing its capabilities. The availability of a working model allows users to develop a clear understanding how the system works and what information is available. Analysis continues, using the adaptable prototype to guarantee that user and analyst are talking about the same thing. As additional specifications are developed, they are incorporated into the model.

After the user is satisfied that specifications have been adequately defined and the prototype has all the desired functions, the next step is to select the most cost-effective techniques for creating the final

product. At this stage, various options need to be carefully and pragmatically evaluated. If the application will be used infrequently or the running time is adequate, it may be most cost-effective to use the prototype without further development. If running or response times are a problem with the prototype, there are two general options to explore: the first is to analyze the application, identify the problem areas and reprogram only the problem segments in a traditional language. In other cases -- high volume, multi-user applications for example -- the prototype will serve as the blueprint from which the final system is programmed. The nature of the application is also a major consideration when selecting programming techniques. For example, decision support systems with their inherent need for flexibility and ad hoc reporting are most effectively handled with data management and information retrieval software such as INFO from Henco, FOCUS from Information builders, NOMAD from National CSS, or DATATRIEVE from Digital Equipment Corportion. In these cases, a refined prototype may be the ultimate system. On the other hand, standard operational applications which are subject to little change and heavy use are candidates for more traditional programming techniques augmented by query languages.

Case Study

An opportunity to apply prototyping occurred recently at Canisius College, a small, private institution in New York State.

A personnel system had been scheduled for development as one part of an integrated, on-line administrative system. When time and resource constraints put the personnel component on "hold" only a few general system requirements had been articulated. Detailed analysis and design were yet to be accomplished.

The author, who is the Director of Planning and Research, had some systems analyst background and a strong motivation to acquire a personnel system since the Planning Office is responsible for all off-campus reporting, including personnel data. After discussing the project with the Computer Center data base administrator, a decision was made to build an interactive prototype which, if successful, could serve as a model for the final system.

Project materials assembled during the initial planning stages were reviewed by the author, who also held several meetings with the Personnel officer, and one meeting with the Academic Vice President to review system needs. Planning office needs were then integrated with those described by the two major users and a rough list of data elements was developed. INFO, a fourth generation tool mentioned above, was available at the College and was used to develop the prototype system.

The first version had only a few input screens, but they were adequate to begin loading demographic, academic, and salary data. During the process of refining system requirements, various data elements were added, deleted, or modified; coding tables were developed and basic reports identified. Because of the relational nature of INFO and the ease of file modification, changes were implemented readily. The definition process was completed after several additional meetings with the users and the prototype is currently in use at the institution.

Two senior computer science students are now completing a reprogramming of the system to meet the standards used for other major Canisius College administrative systems. This structured process includes development of all basic systems in COBOL. All screens are developed with FMS (a product of Digital Equipment Company) and a screen generator

written in-house, and reports are produced with DATATRIEVE. This method of programming standardization produces quality software for multi-user applications in a relatively short time. Additional benefits of the reprogramming will be improved response time, integration of personnel information within the college information system, and an additional level of security for confidential files.

A second project is now in progress. A graduate student intern in the Planning Office is working with the Director of Physical Plant to develop a space inventory system. The prototype system is being developed utilizing DATATRIEVE, another fourth generation product. This project will provide an opportunity to compare the merits of the tools currently available at the College which are suitable for use in the prototyping process. Because of the relatively low volume usage projected for the system it is expected that the refined prototype will be the final system.

Observations and Conclusions

Several observations made during the development of the personnel system are pertinent to the issue of productivity:

Less time was expended in the development process than would be required with traditional methods. The author performed the analysis and prototyping in addition to regular institutional duties. Although the process extended over a three month period, total time devoted to systems development was minimal, totaling approximately 14 people days. Even allowing for the additional time required to develop the COBOL version, total programmer/analyst time devoted to the project will be under 75 days. This compares very favorably with the original estimate of 280 people days needed to develop the system using traditional methods.

Announcing the project as development of a simple prototype removed the personnel officer's fear of making a mistake and overcame her initial

hesitancy in defining system requirements.

The prototype served as a catalyst, increasing useful dialog between analyst and user, which led to rapid system definition.

The prototyping technique is particularly useful when working with users who have little or no experience with computer applications. Novice users quickly respond to the tangible "working model".

Initial concerns that secretarial staff would be confused by working with a changing system, proved unfounded. All modifications were immediately added to a users' guide and staff needed only minimal supervision to adapt to changes.

A number of important benefits can be derived from creating prototypes as part of the system specification process:

1. It involves the user in a productive process, replacing the frustration encountered with traditional project specification methods.
2. Early involvement with a working system develops user sophistication and leads toward development of more comprehensive, satisfactory systems.
3. Users have a working system to use while a final "heavy duty" production model is being programmed and installed.
4. In situations where the system will support light volume usage or the system is to support an ad hoc approach to problem solving, the prototype system becomes the finished system.
5. Design errors are identified and corrected during the prototype stage. Thus system development and maintenance costs are reduced.

The increased programming productivity to be derived from using fourth generation tools is a strong motivation for changing the conventional

systems development process. Even more significant however, is the potential to accelerate the analysis process and at the same time, eliminate many of the problems and frustrations encountered with conventional development efforts. Based upon the experience at Canisius College, the author concludes that the engineering approach to systems development has the potential to significantly increase productivity beyond that achieved with conventional methods. As one of the few approaches to streamlining the analysis and design process, prototyping deserves broader application and further refinement. The appropriate tools are available now. The challenge is to convince computer personnel to use them.

BIBLIOGRAPHY

Bryce, M. Information resource mismanagement. Infosystems, February 1983 pp. 88-92.

Doty, K.E. & Krumrey, A.J. Administrators build their own systems at Loyola University. Cause/Effect, September 1982, pp. 20-23.

Hessinger, P.R. Solving the productivity puzzle: The engineering of information systems. 1982 Spring Seminar Series, Computer Task Group, Inc., Buffalo, New York.

Johnson, R.C. Special report automated software development eliminates application programming. Electronics, June 2, 1982, pp. 129-140.

Lerner, E.J. Programming for nonprogrammers. IEEE Spectrum, August 1982, pp. 34-38.

Martin, J. Application development without programmers. Englewood Cliffs, N.J.: Prentice-Hall, 1982.

McCracken, D.D. The NOMAD approach. Datamation, May 1980, pp. 165-172(a).

McCracken, D.D. Software in the '80's: Perils & Promises. Computerworld, September 17, 1980, pp. 5-10 (b)

Mollen, D. Narrowing the gap. Datamation, May, 1980, pp. 195-198.

Musgrave, E. Relating to data bases. Datamation, Special Issue, 1980, p. 30.

Rhodes, W.L. Jr. The information center extends a helping hand. Infosystems, January 1983, pp. 26-30.

Schultz, B. Programmers seen needing fewer skills. Computerworld, July 28, 1980.

Sheil, B. Power tools for programmers. Datamation, February 1983, pp. 131-136, 142-144.

Stamen, J. & Costello, W. Evaluating database languages. Datamation, May 1981, pp. 118-122.

Thiel, C. DSS means computer-aided management. Infosystems, March 1983, pp. 39-44.

Thiel, C. Programming tools: impacting DP productivity. Infosystems March 1982, pp. 56-60. (a)

Thiel, C. Relational DBMS: what's in a name. Infosystems, September 1982, pp. 52-58. (b)

Wasserman, A.I. & Gutz, S. The future of programming. Communications of the ACM, March 1982, pp. 196-206.